Visible Surface Algorithms for Quadric Patches

Robert Mahl

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF UTAH

UTEC-70-111

VISIBLE SURFACE ALGORITHMS

FOR QUADRIC PATCHES

by

Robert Mahl

December 1970

UTEC-CSc-70-111

This research was supported in part by the University of Utah Computer Science Division and by the Advanced Research Projects Agency of the Department of Defense, monitored by Rome Air Development Center, Griffiss Air Force Base, New York 13440, under contract F30602-70-C-0300.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

ACKNOWLEDGMENTS

The author is specially grateful to Professor David C. Evans who suggested this investigation, directed it, and contributed to some of the ideas of this paper.

Henri Gouraud and Ivan E. Sutherland both helped the author through their encouragement and some insight-giving discussions. Roy Keir and Jeri Panek corrected the final versions of the paper.

VISIBLE SURFACE ALGORITHMS

FOR QUADRIC PATCHES

ABSTRACT

This paper describes two algorithms which find the visible portions of surfaces in a picture of a cluster of three-dimensional quadric patches. A quadric patch is a portion of quadric surface defined by a quadratic equation and by zero, one or several quadratic inequalities. The picture is cut by parallel planes called scan planes; the visibility problem is solved in one scan plane at a time by making "a good guess" as to what is visible according to the visible portions found in the previous scan plane.

The algorithm for intersecting patches works in a time roughly proportional to the number of patches involved (and not to the square of this number as with some previous algorithms).

INTRODUCTION

Numerous studies [2,5] have been devoted to the problem of finding visible portions of surfaces when the surfaces were defined as being planar polygons. Watkins [2] describes one of these algorithms which is quite fast and requires little amount of working storage.

Few successful studies have been made, however, for non-planar surfaces. BE VISION [1] finds the hidden lines when the surfaces are defined by quadratic equations and by quadratic inequalities. Another approach was taken at the University of Utah by Henri Gouraud [3] who approximates Coons' rational cubic surface patches [4] with twisted polygons and then uses Watkins' algorithm in order to find the visible portions of surfaces. In Gouraud's method, the shading of a point of a surface is obtained by interpolating linearly the exact shading value at the four corners of the surrounding polygon, thus obtaining a continuous shading function all over a given surface.

The author appreciates the merits of these previous approaches, but was looking for a method yielding the exact visible contour, intersection lines, and shading functions. To find this, he was forced to limit his investigations to the same type of quadratic surfaces which were used by Weiss [1] in order to limit the computational complexity of the problem to the solution of quartic (4th degree) polynomial equations.

The methods described in the pages following are related to the Watkins' algorithm [2]. They offer two kinds of advantages over the algorithm of BE VISION [1]. First, they yield visible surfaces instead of systematically eliminating hidden lines. Thus they permit the

generation of shaded pictures with only the further effort of computing the shading value along the visible portions of the surfaces. Secondly, they work in an amount of time which is roughly proportional to the number of surfaces fed into the algorithm, rather than the square of this number.

DEFINITION OF QUADRIC PATCHES

Quadric patches are defined in the same way as was used by Weiss [1]. A quadric patch is a set of points obtained by setting a quadratic expression equal to zero and by constraining zero, one or more other quadratic expressions to be either positive or negative.

A quadratic expression is given by:

$$A(x,y,z) = a_1 x^2 + a_2 y^2 + a_3 z^2 + a_4 yz + a_5 zx + a_6 xy + a_7 x + a_8 y + a_9 z + a_{10}$$

The quadratic inequalities are used, in particular, to define the boundaries of the quadric patches. For example, suppose that we want to define an infinitely long cylinder in the z direction, centered at (0,0) and of radius 1; its equation would be:

$$(1) \quad x^2 + y^2 - 1 = 0$$

If we were interested in the portion of the cylinder truncated by the planes $z = 1$ and $z = -1$, we would impose the constraints

$$(2) \quad z + 1 \leq 0$$

$$(3) \quad z - 1 \geq 0$$

If, moreover, we want to bore a hole through this cylinder, we may impose the additional constraint:

$$(4) \quad x^2 + z^2 - .25 \geq 0$$

Equations (1) to (4) define the resulting patch, shown in Figure 1.

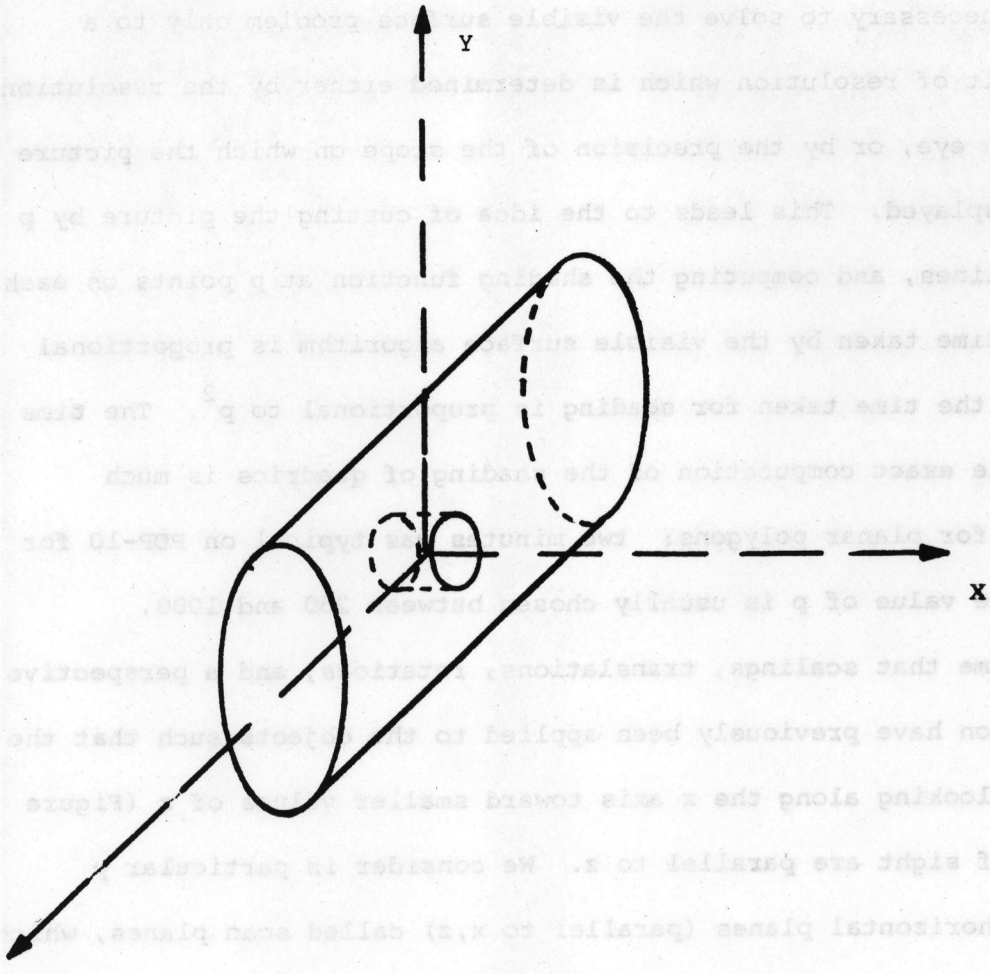

Figure 1: Cylinder with hole.

SCAN LINE TECHNIQUES

It is necessary to solve the visible surface problem only to a certain limit of resolution which is determined either by the resolution of the human eye, or by the precision of the scope on which the picture is to be displayed. This leads to the idea of cutting the picture by p horizontal lines, and computing the shading function at p points on each line. The time taken by the visible surface algorithm is proportional to p, while the time taken for shading is proportional to p^2. The time taken for the exact computation of the shading of quadrics is much larger than for planar polygons: two minutes was typical on PDP-10 for p = 500. The value of p is usually chosen between 250 and 1000.

We assume that scalings, translations, rotations, and a perspective transformation have previously been applied to the objects such that the observer is looking along the z axis toward smaller values of z (Figure 2). Lines of sight are parallel to z. We consider in particular p equidistant horizontal planes (parallel to x,z) called scan planes, which intersect the transformed objects. These scan planes are projected onto the image plane as p equidistant scan lines.

We solve the visible-surface problem for a given scan line by cutting the object surfaces by the scan plane. The intersections of quadric patches with this plane are portions of conics. The problem is then solved in the current scan plane, and the values of the shading function are computed along the scan line for the visible portions of conics. This method of computing visibility is made fast because we keep track of the visible portions in the previous scan plane, thus reducing

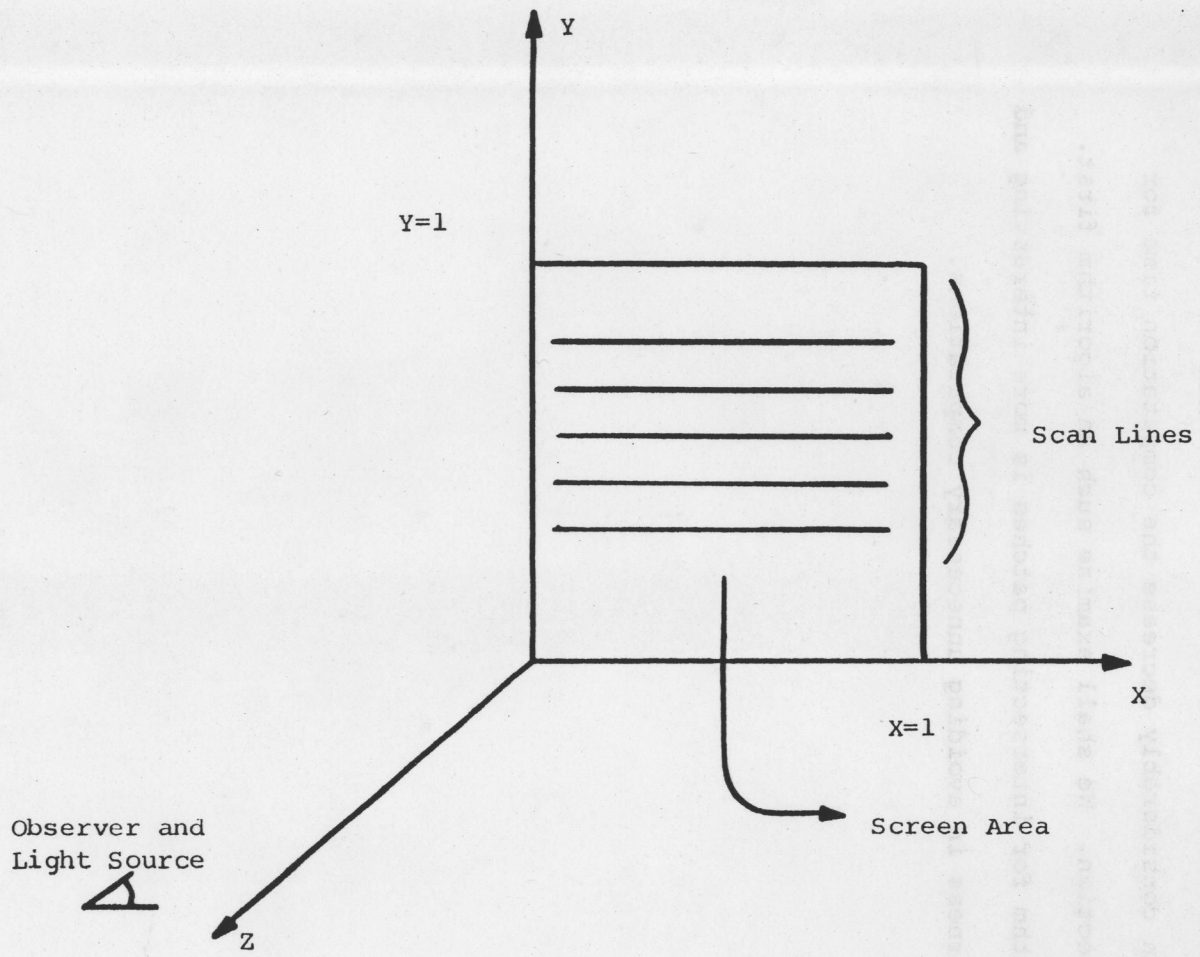

Figure 2: Observer's relation to scan lines.

considerably the number of comparisons of portions of conics necessary in order to determine which one is in front of the others.

We are now going to examine two visible surface algorithms: one for non-intersecting quadric patches, and another one for possibly intersecting quadric patches. If patches are known not to intersect one another, one can considerably decrease the computation time for visible surface detection. We shall examine such an algorithm first. However, the algorithm for intersecting patches is more interesting and involves some cleverness in avoiding unnecessary computations.

COMPUTATION OF THE SPECIAL POINTS OF EACH PATCH IN A GIVEN SCAN PLANE

The intersection of a quadric by a scan plane is a conic, perhaps degenerate. For the observer situated at infinity in the z direction, a conic has two portions: the closest portion and the furthest portion. If a conic has two different points for the same given x coordinate value, the one with the greatest z value belongs to the closest portion, the other one to the furthest portion.

Spceial points of a conic are of two kinds: apparent contour points (x_1 and x_4 in Figure 3) where the normal is perpendicular to the z axis, and points of intersection of the conic with its boundaries (x_2 and x_3). These points are clipped outside the [0,1] interval (points outside this interval are eliminated as being off the screen). They are then ordered according to increasing x values. For each interval $[x_i, x_{i+1}]$, the visible portion of the patch is remembered. For instance in Figure 3 these intervals are:

Interval	Visibility
$[x_0 = 0, x_1]$	0 (not visible)
$[x_1, x_2]$	1 (closest portion)
$[x_2, x_3]$	2 (furthest portion)
$[x_3, x_4]$	1 (closest portion)
$[x_4, x_5 = 1]$	0 (not visible)

In the first phase of the algorithm, an explicit representation of the conic z = f(x) is also obtained; it will be used to do the depth computations in the next section and later to obtain the shading values.

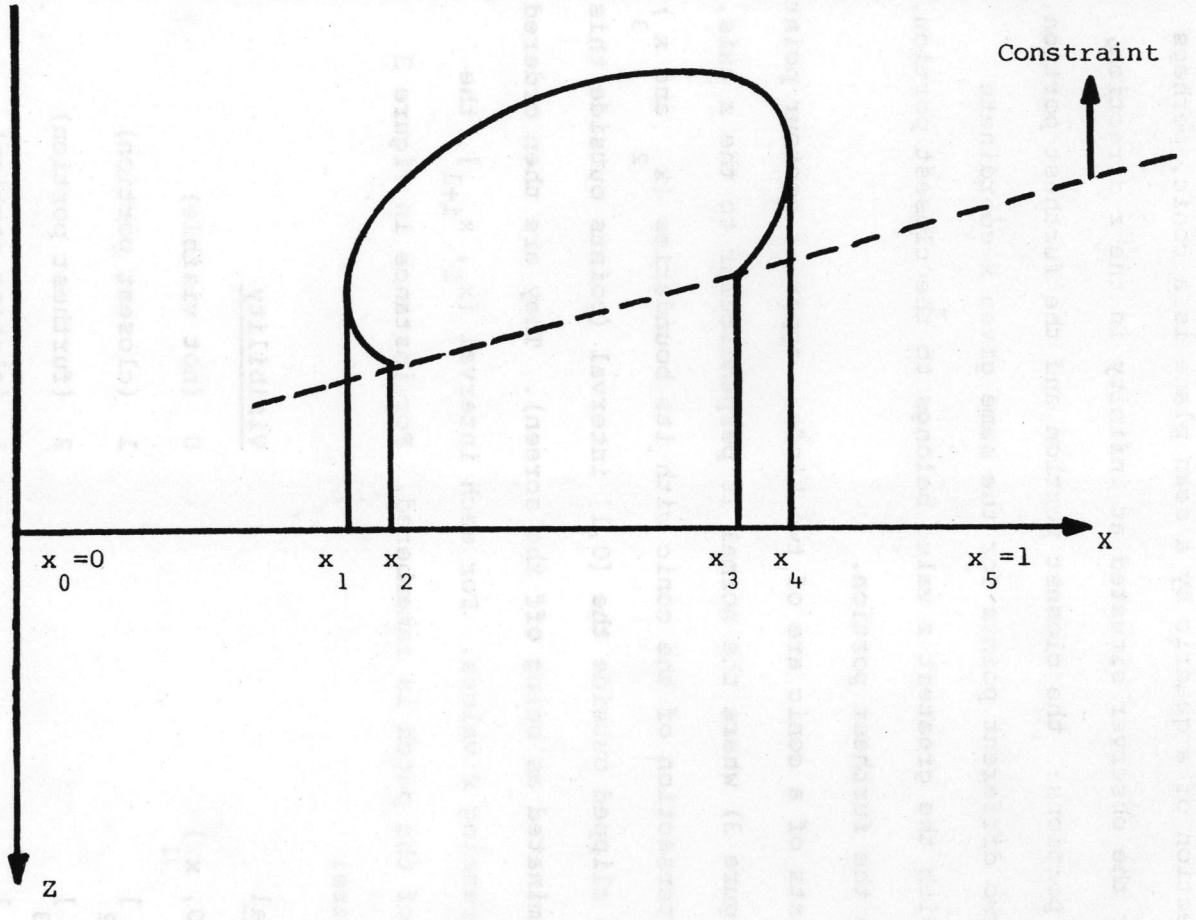

Figure 3: Intersection of the quadric and the scan plane "Y=CST" showing the different "special points".

THE ALGORITHM FOR NON-INTERSECTING PATCHES

We have determined, in a scan plane, the visibility of each patch for the observer if all other patches were removed. Now we must synthesize all this information.

The first step is to merge together the special points of all the patches into one large list which is ordered by increasing x coordinates. This list will then be scanned. For each interval $[x_i, x_{i+1}]$ we shall determine which is the visible patch, if any. Of course at that moment the visibility in the $[x_{i-1}, x_i]$ interval has already been determined, except for i = 0. Thus:

--if i = 0 or if x_i is a special point for the patch which was visible in the $[x_{i-1}, x_i]$ interval, we have to compare all patches in the $[x_i, x_{i+1}]$ interval. This is achieved by computing the depth of each patch (z coordinate) in the interior of the interval, for instance at the midpoint $x = \frac{x_i + x_{i+1}}{2}$. Those depths are then sorted, and the patch with the greatest z is declared visible.

--if x_1 is a special point for patch P where a new portion of patch P might become visible, and if patch P' was found to be visible in the interval $[x_{i-1}, x_i]$ with $P' \neq P$, then P and P' must be compared, but not to any other patch. Of P and P', the one closest to the observer is declared visible. If no patch was visible in the previous interval, then patch P automatically becomes visible.

This algorithm can be slightly improved by observing if a patch A was in front of another patch A' in an interval $[x_{i-1}, x_i]$ and if x_i is not a special point for either of these patches, the patch A will still cover A' in interval $[x_i + x_{i+1}]$. Thus we can avoid unnecessary depth

computations by maintaining depth-ordered lists of some of the patches while we are scanning (increasing x). Whenever a special point is found for a patch, this patch is deleted from the depth-ordered lists, except if it is the patch which will be seen in the next x interval, in which case it is placed in front of all lists. The usefulness of these lists arises when we arrive at a point x_i which is a special point for the patch which was seen in the $[x_{i-1}, x_i]$ interval, thus requiring all patches to be compared. However, patches which are below the top of any one of our depth-ordered lists will certainly not be seen because they are at least hidden by the patch which is at the top of that list. Those patches which cannot be eliminated in this way will need to be compared and will be depth-sorted into a next list. A depth-ordered list is deleted whenever it is reduced to only one element. Because a given patch may exist in several of these lists at a time, it is desirable to have all occurrences of the same patch themselves list linked in order to be able to delete them rapidly if we arrive at a special point for this patch.

A VISIBLE SURFACE ALGORITHM FOR INTERSECTING PATCHES

If patches may intersect, we cannot compare them on an interval by computing their depths at some arbitrary point in this interval. The correct way to compare two patches on an interval is:

1. To compute their intersections.
2. If there is any intersection in the interval, to subdivide the interval (creating n + 1 subintervals if there are n intersections) so that the patches do not intersect on each interval.

and 3. To compare depths at any arbitrary point within each interval in which the patches are now known not to intersect.

If n patches are involved, this method would require about $n^2/4$ comparisons in order to determine which patch is in front of the others on an interval. The computation of the intersections of two conics is an expensive operation which requires around 3 milliseconds of computation time on the PDP-10 (on which these algorithms were implemented by the author). Most of this time is devoted to solving a quartic equation. Thus it becomes critical to reduce the number of comparisons of quadrics and of computations of intersections. This can be done in several ways:

1. When the intersection of two conics has been determined, they must be stored somewhere in order to avoid having to recompute the same intersections on the same scan line. If the numbers k and k' identify two conics, the intersections of k and k' are stores in a hashcoded array. The access function in this array (or hash function) H(k,k') has to be symmetric: H(k,k') = H(k,k'), because

the intersections of k and k' are identical to the intersections of k' and k.

2. In order to minimize the number of comparisons, we use a "good guess" of which patch will be visible in a given interval. This guess is obtained by remembering the "sample points" and the "sample segments" (see also reference [2]) of the previous scan line. A sample segment is a segment of the previous scan line which corresponds to a visible portion of a conic. A sample point is the extremity of a sample segment. We use these points of the previous scan line in order to get sample points belonging to the same patches as - and being as close as possible to - sample points of the previous scan line.

On each sample interval, we have to compare the probably visible patch with other possibly visible patches on that interval. Thus if there are n patches and if the visible segments in the previous scan plane corresponded to v different patches ($v \leq n$), then the number of comparisons will be less than $(n-1)v$, if the good guesses were right, which is usually the case. Moreover, if there are on the average q possibly visible patches behind each of the v probably visible patches, then the total number of comparisons required is qv. Thus the compute time of the algorithm is proportional to the "visible complexity" v of the picture.

Of course, if the good guess is wrong, the sample segment eventually has to be divided and more comparisons are needed:

3. If A is in front of B and B is in front of C, then A is in front of C. This might permit us to avoid having to compute the intersection of A and C in some cases where the intersections of A and B and of B and C were

already computed previously on the scan line. The interest of this "trick" comes from the fact that it is much more expensive to compute the intersections of two conics than just to compute their depths for some value of x.

4. Another improvement which was suggested to the author, but not implemented in his computer program, consists of using the values of the coordinates of the intersections of two patches in the previous scan plane, as initial approximations of their values in the current scan plane, and to refine the approximation for the new scan plane by applying a two-dimensional Newton method.

SHADING THE CURVED SURFACES

For an observer looking from a distance d in the z direction, and if the light source coincides with the observer, an acceptable and pleasing shading value is given by:

$$S = \frac{\cos^2\alpha}{(d-z)^2}$$

At the point of the surface of coordinates (x,y,z), α is the angle between the normal to the surface and the direction of the observer (z direction); we have:

$$\cos^2\alpha = \frac{n_z^2}{n_x^2 + n_y^2 + n_z^2}$$

where (n_x, n_y, n_z) are the components of the normal, given by:

$$n_x = 2a_1x + a_6y + a_5z$$

$$n_y = 2a_2y + a_4z + a_6x$$

$$n_z = 2a_3z + a_5x + a_4y$$

Depending on the type of quadric, z is computed either by solving a quadratic equation, or by computing the ratio of the value of a second-order polynomial to the value of a first-order polynomial. The fact that the components of the normal and the value of z have to be computed for hundreds of thousands of points for each picture makes the cost of shading generally much more expensive than the cost of visible surface detection.

THE DATA STRUCTURE

An interesting by-product of this investigation is a system which allows one to create and modify patches and pictures interactively.

The picture designer can model a data structure. This structure can be viewed as an oriented graph without loops, in which appear only the names of objects and the names of transformations, and not the actual coefficients of the transformation matrices or of the quadratic equations. The same name may appear in several places in the structure, and these occurrences do not have to be changed when the attributes of the name are to be modified.

The nodes of the graph are occupied by the names of three types of objects:

- subpictures
- patches

and - quadratic patches

The graph edges are occupied by the names of transformations and, in some cases, by a special attribute which is greater, less than, or equal to zero. This special attribute appears only if the vertex joins the name of a patch to the name of a quadratic equation.

A language has been designed and implemented on the PDP-10 to allow one to define transformations and quadric patches, and to create or delete objects and father-son relationships. Examples of sentences of this language are:

TRANS total AS PROD ROTX 0 CENTR .5 .5 .5 TRZ .4 END

which defines the transformation TOTAL as the product of a centered rotation and of a translation;

IN sub INCL obj BY total

creates a father-to-son relationship between the subpicture "sub" (father), and the object "obj" to which is applied the transformation referenced by the name "total";

DISP sub BY persp

displays object "sub", transformed by the transformation "persp".

Figure 4 shows the set of commands which were used in order to define the structure of the rocket displayed in Figure 5. Other examples of pictures produced by this algorithm are shown in Figures 6 and 7.

```
MODE NOINT
QUADR CYL AS EQU 1 0 1 0 0 0 0 0 0 -1
QUADR CONE AS EQU 1 -1 1
QUADR PLANE AS EQU 0 0 0 0 0 0 0 0  1
TRANS TC1 AS PROD SCALE .1 TRX .5 END
TRANS TC2 AS PROD SCY 2 ROTX 180 TR .5 .7 END
TRANS TC3 AS PROD SCALE .05 TRX .5 END
TRANS TC4 AS PROD SCY 2 ROTX 180 TR .5 .8 END
TRANS TBOTT AS ROTX -90
TRANS TOPC1 AS TBOTT CENTR 0 .5
TRANS TOPC2 AS TBOTT CENTR 0 .6
TRANS TOPC3 AS TBOTT CENTR 0 .7
TRANS TOPC4 AS TBOTT CENTR 0 .8
PATCH C1 IN C1 INCL CYL BY TC1
                INCL PLANE BY TBOTT POS
                INCL PLANE BY TOPC1 NEG
PATCH C2 IN C2 INCL CONE BY TC2
                INCL PLANE BY TOPC1 POS
                INCL PLANE BY TOPC2 NEG
PATCH C3 IN C3 INCL CYL BY TC3
                INCL PLANE BY TOPC2 POS
                INCL PLANE BY TOPC3 NEG
PATCH C4 IN C4 INCL CONE BY TC4
                INCL PLANE BY TOPC3 POS
                INCL PLANE BY TOPC4 NEG
TRANS OH1 AS PROD ROTY 90 TRX .5 END
TRANS OH2 AS PROD OH1 ROTZ 225 CENTR .5 .3 END
PATCH A3 IN A3 INCL PLANE
                INCL PLANE BY TBOTT POS
                INCL PLANE BY OH1 POS
                INCL PLANE BY OH2 POS
                INCL CYL BY TC1 POS
TRANS TA2 AS ROTY -90 CENTR .5
SUBP A2 IN A2 INCL A3 BY TA2
SUBP A1 IN A1 INCL A2 BY TA2
SUBP A4 IN A4 INCL A1 BY TA2
SUBP TOTAL IN TOTAL INCL A1 INCL A2
   INCL A3 INCL A4
   INCL C1 INCL C2
   INCL C3 INCL C4
SCAN 500
TRANS TTT AS PROD ROTX 30 ROTY 40 ROTZ 50 END CENTR .5 .5
TRANS TTT AS PROD TTT TR -.1 .1 END
MODE SHADE
SHADE 1 J 1500
SUBP ROTA IN ROTA INCL TOTAL BY TTT
DISP ROTA
TTY
```

Figure 4

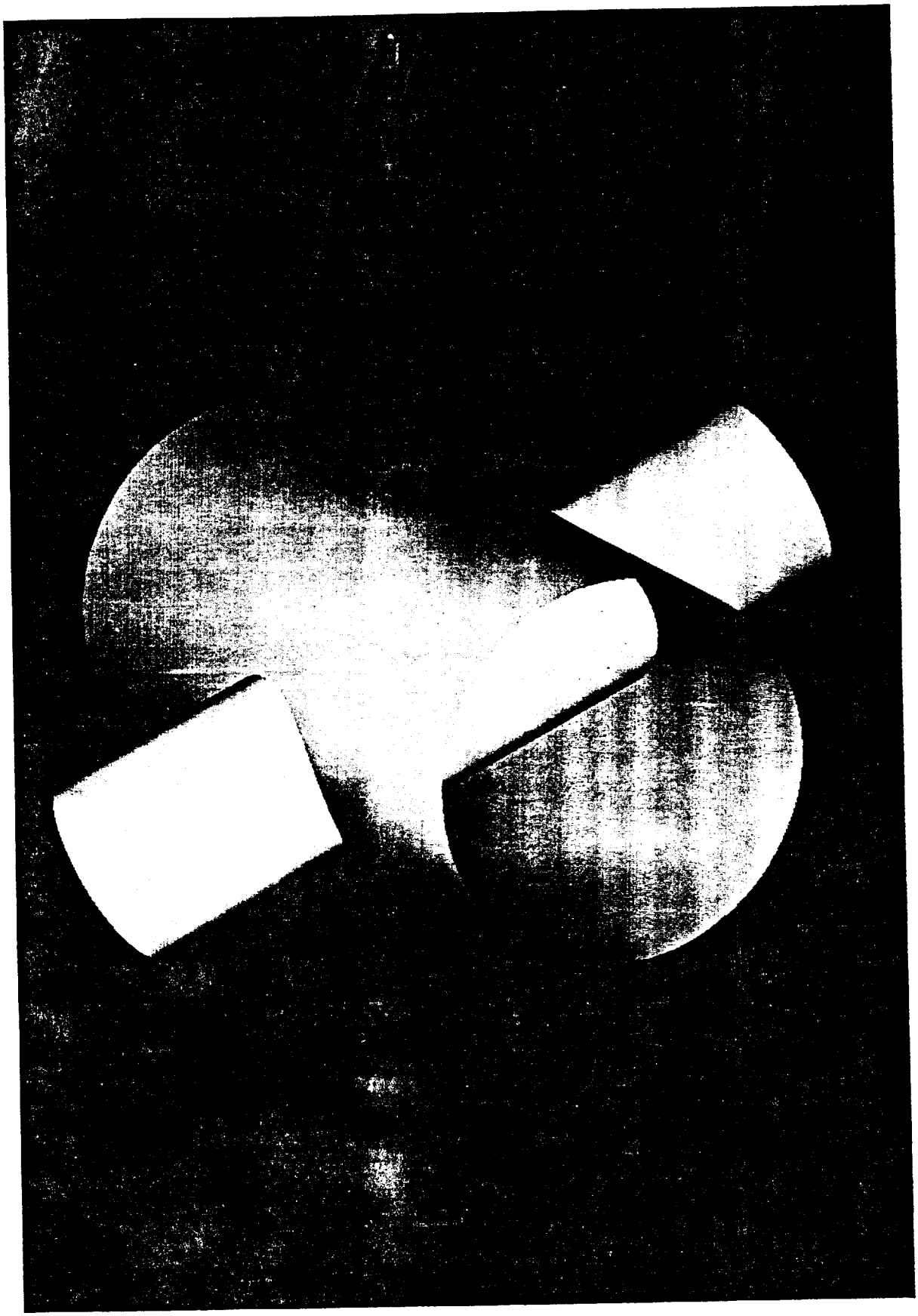

Figure 5: Two intersecting cylinders.

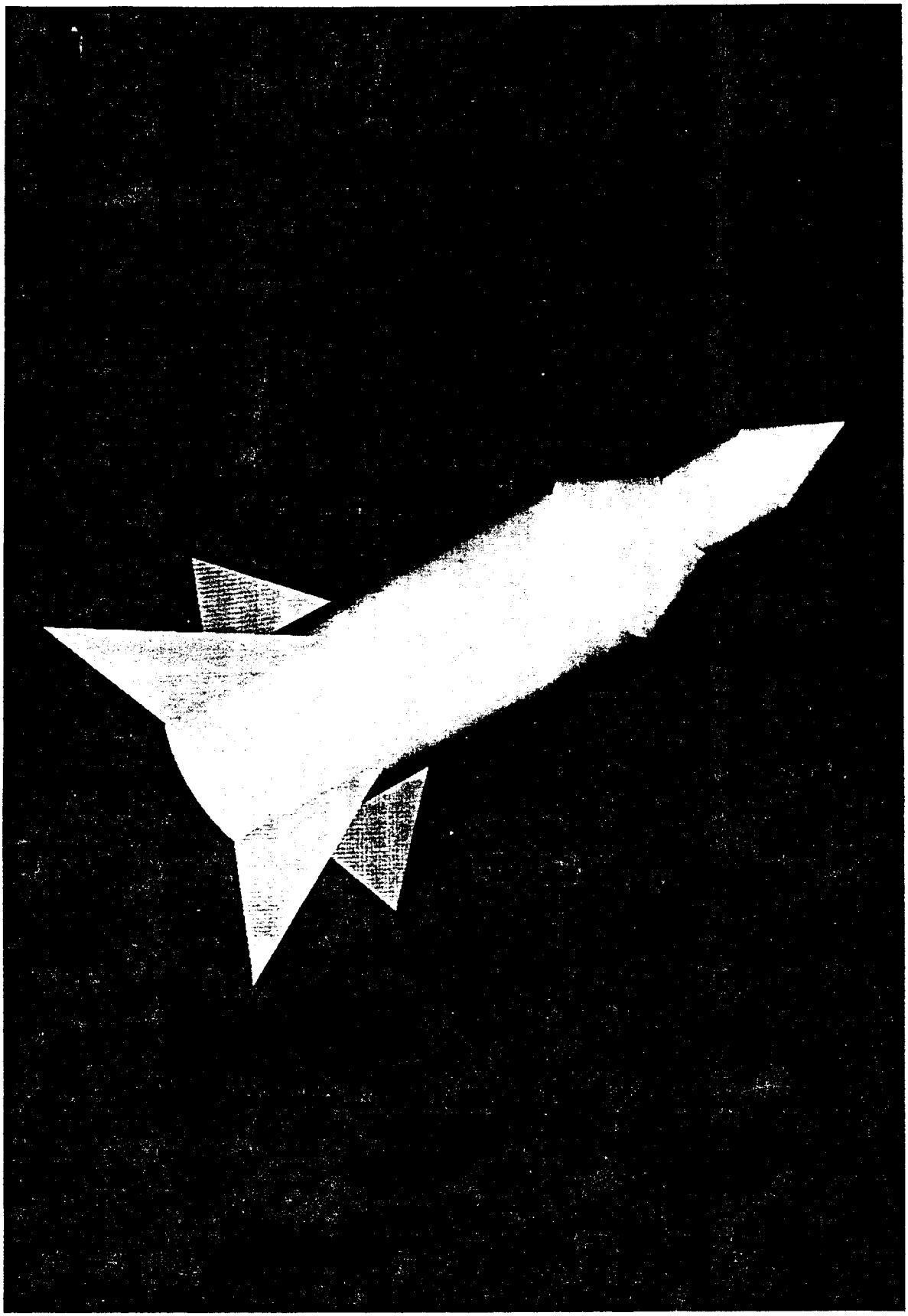

Figure 6: "Rocket" made from portions of cylinder cones and planes.

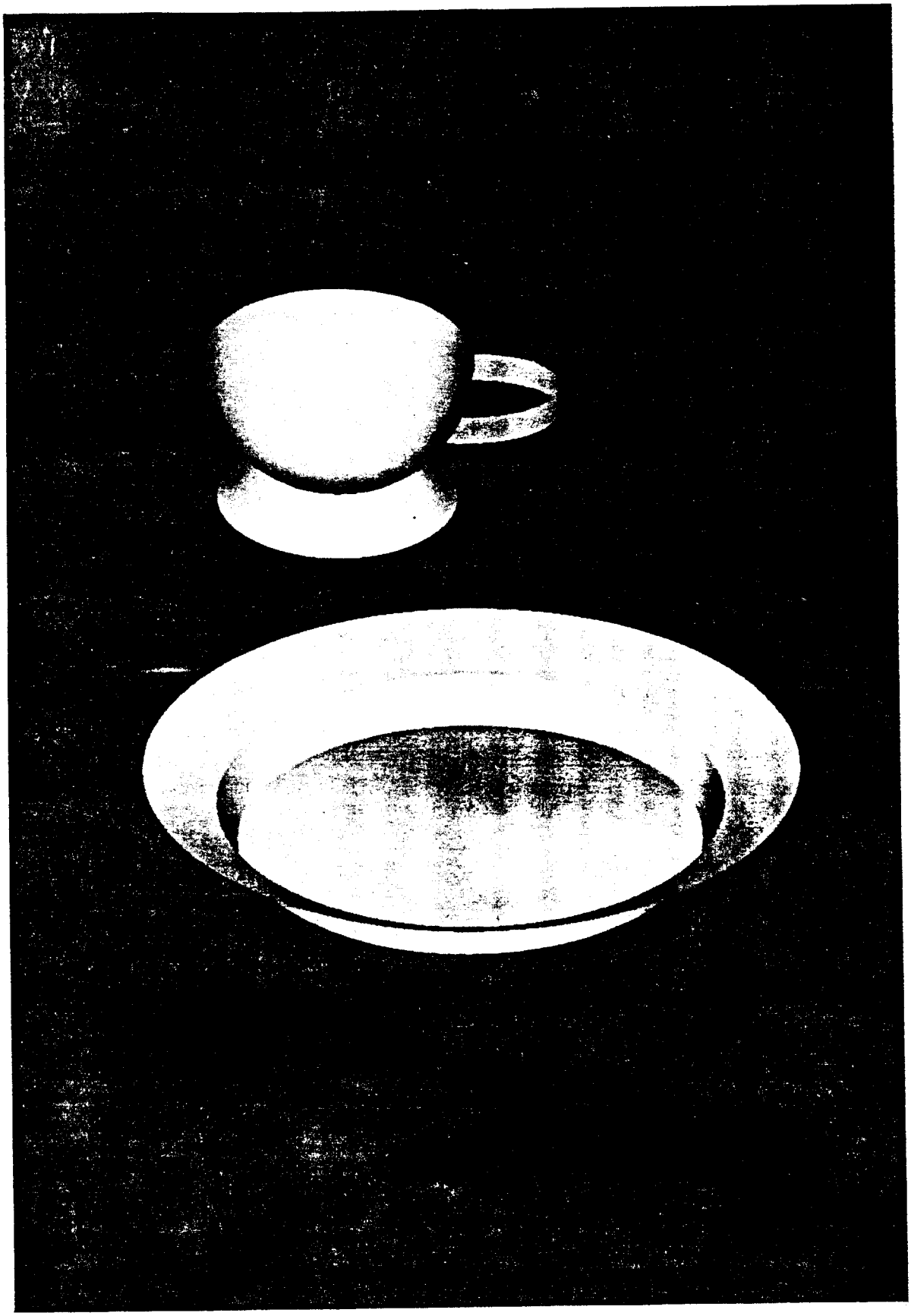

Figure 7: Cup and Saucer

CONCLUSION

Algorithms have been presented which determine the visible portions of quadric patches. The algorithm for patches known not to intersect works much faster than the other algorithm. In both cases, however, the computation of the precise shading value at each point of the picture is in general the most expensive process, even though we assumed that the light source and the observer were at the same place.

More complicated classes of surfaces still have to be investigated, but this will probably require more computational power. It would also be interesting to have an algorithm which (like Watkins' algorithm) would work very fast if only planar polygons are involved, but which could also handle quadric surfaces at the cost of a smooth degradation of performance.

REFERENCES

1. Weiss, Ruth A., BE VISION, A Package of IBM 7090 FORTRAN Programs to Draw Orthographic Views of Combinations of Plane and Quadric Surfaces, *Journal of the ACM*, 13, 2, April 1966, pp. 194-204.

2. Watkins, G.S., A Real-time Visible Surface Algorithm. Ph.D. dissertation, *Technical Report*, UTEC-CSc-70-101, Computer Science Division, University of Utah, July 1970.

3. Gouraud, H., Continuous Shading of Curved Surfaces, *Technical Report*, Computer Science Division, University of Utah, July 1970.

4. Coons, S. A., Surfaces for Computer-aided Design of Spaceforms, *Project-MAC Report*, MAC-TR-41, June 1967.

5. Comba, P. G., A Procedure for Detecting Intersections of Three-dimensional Objects, *IBM Report No. 39020*, New York Scientific Center, January 1967.